CHECKPOINT 3

¿Cuáles son los tipos de Datos en Python?

Los principales tipos de Datos en Python son 9.

- Strings: Almacenan secuencias de bytes. Desde un nombre u oración hasta documentos HTML completos.
- Booleans: dan una respuesta de True o False
- Numbers: almacenan datos numéricos, fracciones etc.
- Bytes and byte arrays: Datos más complejos y avanzados
- None: ningún dato, cuando quieres poner una variable, pero aún no tiene un valor definido.

Dentro de Python también se podrán gestionar librerías de colecciones; list, tuplets, sets y dictionaries.

¿Qué tipo de convención de nomenclatura deberíamos utilizar para las variables en Python?

Las variables son contenedores para diferentes elementos de datos; cálculos, números, oraciones, párrafos… En Python, se utilizan prácticas estándares para nombrar las variables y que cualquiera que utilice nuestro código pueda entenderlo. Dentro, habrá diferentes *Data Types*. Seguiremos las guías de estilo de PEP 8, algunos de los ejemplo para las variables son los siguientes.

nombre = 'Ane'

En este caso la variable se llama nombre y dentro de esta (string) está almacenado el nombre Ane

Cuando queremos poner varias palabras en una variable usaremos el *snake case*:

student_name = 'Ane'

Por último, para ver el resultado, deberemos de hacer lo siguiente:

print(nombre)

print(student_name)

También habrá nombres o letras que queramos evitar para que no haya confusiones como con la letra L minúscula o con el cero y la letra O y también tendremos que ser descriptivos con los nombres de las variables, para saber qué tipo de dato es el que almacenan.

¿Qué es un Heredoc en Python?

Un *Heredoc* simplemente son *multiline strings*. Se utilizan *"""* o *''',* y permite escribir textos más extensos en diferentes líneas.

mensaje = """Hola,

Este es un mensaje heredoc.

Contiene varias líneas de strings.

"""

¿Qué es una interpolación de cadenas?

La interpolación de cadenas nos da la habilidad de procesar el código Python dentro de las strings. Hay elementos que se repetirán y otros que serán dinámicos, por ejemplo, el email que recibiría cada suscriptor de una newsletter con su nombre.

nombre= 'Ane'

mensaje= f 'Hola {nombre}, este es tu primer email.'

print(mensaje)

Hola Ane, este es tu primer email.

¿Cuándo deberíamos usar comentarios en Python?

Depende de la situación deberemos de usarlos o no ya que a veces pueden ser beneficiosos pero otras veces pueden llegar a dar problemas y entorpecer el desarrollo de la aplicación. Por ejemplo, si cambiamos algo en el código y no actualizamos los comentarios, los comentarios se vuelven inservibles. Por eso, no se recomienda usar comentarios para explicar el funcionamiento o para añadir instrucciones.

Si se llama a un componente de la forma correcta, los comentarios no suelen ser necesarios. Los comentarios deberían de usarse por ejemplo para organizar nuestro código o para propósitos referenciales.

¿Cuáles son las diferencias entre aplicaciones monolíticas y de microservicios?

Para elegir el tipo de sistema que necesitas hay muchas cosas que tener en cuenta: que tipo de app necesitas construir, cuanto es el tiempo que tienes para hacerlo, cuál es tu presupuesto y cuál es tu experiencia.

Monolíticas	Microservice
La estructura monolítica es una configuración utilizada para los sistemas tradicionales del lado del servidor. La función de todo el sistema se basa en una sola aplicación, las diferentes partes dentro de esta se comunican entre sí. Es más fácil de desarrollar y se puede crear una aplicación con características básicas y luego ampliarla con el tiempo. Pueden ser más rápidas porque no tienen el requisito de comunicarse a través de las API. Mantenerlas puede ser un reto si no están bien desarrolladas ya que hacer un solo cambio puede tener efecto en alguna otra parte de la aplicación.	Es una arquitectura en la que cada característica es su propia aplicación. Para poder conectar las aplicaciones hay que saber sobre APIs. Las aplicaciones no dependen unas de otras, sino que cada una es independiente por lo que por ejemplo si una se cae, no se nos caería toda la web, sino que solo esa parte dejaría de funcionar. Algunos de los beneficios son que por ejemplo al trabajar en equipo que cada uno puede dedicarse única y exclusivamente a una aplicación. También nos da la capacidad de ampliar los componente individualmente sin tener que hacerlo en toda la app completa. Este tipo de aplicaciones se utilizan en sistemas grandes y más complejos, ya que se tarda más en construir y requiere más recursos